```
// Compliance.cpp
// Copyright (c) 2003. Zone Labs, Inc. All Rights Reserved.
/**********************************************************************/
/**                    Zone Labs Monitor                             **/
/**********************************************************************/
/*
 *  Compliance.cpp (Integrity2.lib, VSMon.exe)
 *
 *  Client-side cooperative enforcment; TV-independent. See
 *  cooperative enforcement spec for details.
 *  Parses expressions in the compliance rule format.
 *
 *
 * ---------------------------------------------------------------------
 * Compliance Rule Syntax
And/or may only apply to expression operators.
<xs:simpleType name="complianceOperator">
   <xs:restriction base="xs:string">
      <xs:enumeration value="eq"/>
      <xs:enumeration value="ne"/>
      <xs:enumeration value="lt"/>
      <xs:enumeration value="gt"/>
      <xs:enumeration value="between"/>
      <xs:enumeration value="and"/>
      <xs:enumeration value="or"/>
   </xs:restriction>
</xs:simpleType>
<xs:simpleType name="complianceStatus">
   <xs:restriction base="xs:decimal">
      <xs:minInclusive value="0"/>
      <xs:maxInclusive value="1"/>
   </xs:restriction>
</xs:simpleType>
Specifying "not" in an expression negates its result.
<xs:simpleType name="complianceUnaryOperator">
   <xs:restriction base="xs:string">
      <xs:enumeration value="not"/>
   </xs:restriction>
</xs:simpleType>
<xs:simpleType name="complianceOperandType">
   <xs:restriction base="xs:string">
      <xs:enumeration value="int"/>
      <xs:enumeration value="date"/>
      <xs:enumeration value="version"/>
      <xs:enumeration value="string"/>
      <xs:enumeration value="expression"/>
   </xs:restriction>
</xs:simpleType>
The "value" attribute of an operand is evaluated according to the type of the enclosing expression.  If it is "expression",
then (and only
```

then!) are the inherited complianceExpression attributes and subelements expected. If the "provider" attribute is present, then the value string represents the name of the attribute of the specified provider.

```
<xs:complexType name="complianceOperand">
   <xs:attribute name="provider" type="xs:string" use="optional" />
   <xs:attribute name="value" type="xs:string" use="required" /> </xs:complexType>
```

An expression may have either of the following:

- Operand[s], whose type is taken as the "type" attribute of the expression.
- One or more subexpressions, only if the "type" of the expression is "expression".
  In this case, the operator may only be one of the logical operators (or, and, etc.)

```
<xs:complexType name="complianceExpression" minOccurs="1">
   <xs:attribute name="type" type="complianceOperandType" use="required"/>
   <xs:attribute name="unaryOp" type="complianceUnaryOperator" use="optional"/>
   <xs:attribute name="operator" type="complianceOperator" use="optional"/>
   <xs:sequence>
      <xs:element name="operand1" type="complianceOperand" use="optional" maxOccurs="1"/>
      <xs:element name="operand2" type="complianceOperand" use="optional" maxOccurs="1"/>
      <xs:element name="expression" type="complianceExpression" use="optional" />
   </xs:sequence>
</xs:complexType>
<xs:complexType name="complianceRule" minOccurs="0">
   <xs:attribute name="status" type="complianceStatus" use="optional" />
   <xs:attribute name="compliant" type="xs:string" use="optional" />
   <xs:attribute name="message" type="xs:string" use="optional" />
   <xs:attribute name="messageUrl" type="xs:anyURI" use="optional" />
   <xs:attribute name="task" type="xs:anyURI" use="optional" />
   <xs:sequence>
      <xs:element name="expression" type="complianceExpression" use="required" minOccurs="1"/>
   </xs:sequence>
</xs:complexType>
```

Examples:

```
<complianceRule status="0" compliance="ZL_IAVERSION" message="Old client version." >
   <expression operator="ge" operandtype="version">
      <operand1 value="4.0.0.1"/>
      <operand2 provider="zonelabs" value="clientVersion"/>
   </expression>
<complianceRule>
```

An example using subexpressions. Also note the use of an ACE rule result in the third operand. An ACE rule called "napster" is expected to be present. The value of an ACE rule is 1 for compliant, 0 for noncompliant.

```
<complianceRule compliance="ZL_IAVERSION" status="0" message="Not compliant." >
   <expression type="version" operator="ge" message="Old client version." >
      <operand1 value="4.0.0.1"/>
      <operand2 type="attribute" provider="zonelabs" value="clientVersion"/>
   </expression>
   <expression type="expression" operator="or" >
      <expression type="date" operator="ge" >
         <operand1 value="03/04/2002"/>
         <operand2 provider="Symantec.nav" value="datDate"/>
      </expression>
      <expression type="date" operator="ge" >
         <operand1 value="03/12/2002"/>
```

```
        <operand2 provider="Norton.nav" value="datDate"/>
      </expression>
    </expression>
    <expression type="int" operator="gt">
      <operand1 provider="zonelabs" value="ACE.napster"/>
      <operand2 value="1"/>
    </expression>
<complianceRule>
Conrad's example:
<complianceRule status="ZL_IAVERSION" message="Old client version." >
  <expression operator="ge" operandtype="version">
    <operand1 provider="zonelabs" value="engineVersion"/>
    <operand2 provider="zonelabs" value="clientVersion"/>
  </expression>
</complianceRule>
An example which provides a message for a failed ACE rule:
<complianceRule compliance="ZL_IAVERSION" status="0" message="Not allowed to run Napster." >
    <expression type="int" operator="eq">
      <operand1 value="1"/>
      <operand2 provider="zonelabs" value="ACE.napster"/>
    </expression>
</complianceRule>
 *
 *
 * Compliance property list.  Remember: case insensitive.
 * ZoneLabs provider:
 *     engineVersion
 *     clientVersion
 *     clientName
 *     missedHeartbeats
 *     noncompliantHeartbeats
 *
 * "System" provider property list.
 *     osName              string     Windows NT/2000/XP, Windows 9x/ME, Windows NT, Windows XP, etc.
 *     registry.value      string     Matches exact value
 *     registry.exists     int        1 if entry exists, 0 otherwise
 *     file.exists         int        1 if named file is present
 *     file.running        int        1 if named file is running
 *     file.version        version
 *     file.datetime       datetime
 *     file.checksum       string
 *
 * registry.value.HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Run
 * registry.exists.HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Run\QuickTime Task
 *
 * file.running.iexplorer.exe
 * file.version.%windir%\system32\zonelabs.vsmon.exe
 * file.version.%windir%\system32\zonelabs.vsmon.exe
 * file.checksum.%windir%\system32\zonelabs.vsmon.exe
 * ------------------------------------------------------------------------
```

```
 */
#include "VSNetLibPCH.h"
#include "Compliance.h"
#include "systemprovider.h"
LOG_INIT;
ComplianceOperator::Operator ComplianceOperator::OperatorFromString( const char *szOp )
{
   if ( ! szOp && szOp[0] )
      return err;
   struct{ Operator op; char *szOp; } OpTable[] =
   {
      { eq,        "eq" },
      { ne,        "ne" },
      { lt,        "lt" },
      { gt,        "gt" },
      { le,        "le" },
      { ge,        "ge" },
      { youngerDays, "youngerDays" },
      { and,       "and" },
      { or,        "or" },
      { err,       "\0" }
   };
   int i = 0;
   while ( OpTable[i].op != err )
   {
      if ( ! stricmp( szOp, OpTable[i].szOp ) )
         break;
      i++;
   }
   return OpTable[i].op;
}
ComplianceOperandType::Type ComplianceOperandType::TypeFromString( const char *szType )
{
   if ( ! szType && szType[0] )
      return err;
   struct{ Type tp; char *szType; } TypeTable[] =
   {
      { Int,        "Int" },
      { date,       "date" },
      { version,    "version" },
      { string,     "string" },
      { expression, "expression" },
      { err,        "\0" }
   };
   int i = 0;
   while ( TypeTable[i].tp != err )
   {
      if ( ! stricmp( szType, TypeTable[i].szType ) )
         break;
      i++;
```

```
    }
    return TypeTable[i].tp;
}
//---------------------------------------------------------------------
//---------------------------------------------------------------------
DATETIME ComplianceExpression::MakeComparisonDate( ComplianceOperator::Operator exprOp,
                                    DATETIME                Op1,
                                    int                     Op2 )
{
    if ( exprOp == youngerDays )
    {
        return DateTimeAddDays( GetCurrentDateTime(), - Op2 );  // compare op1 > today - days
    }
    else
        return 0;   // Error
}
ComplianceResult::expResult ComplianceExpression::Evaluate( ProviderReporterList &Providers,
                                        ProviderReportList   &ProviderReports )
{
   return EvaluateSubExpr( m_expression, Providers, ProviderReports );
}
ComplianceExpression::expResult ComplianceExpression::CalcAndCompareOperands(
ComplianceOperandType::Type  exprType,
                                                ComplianceOperator::Operator exprOp,
                                                const xmlstring             &Expression,
                                                ProviderReporterList        &Providers,
                                                ProviderReportList          &ProviderReports )
{
    xmlstring operand1, operand2;
    Expression.GetXMLTag( "<operand1>", operand1, 0 );
    Expression.GetXMLTag( "<operand2>", operand2, 0 );
    faststring val1( operand1.GetParameter( "value" ) );
    faststring val2( operand2.GetParameter( "value" ) );
    fastistring provider1( operand1.GetParameter( "provider" ) );
    fastistring provider2( operand2.GetParameter( "provider" ) );
    expResult ResultValue = eErr;
    faststring propName1;
    faststring propName2;
    switch( exprType )
    {
        case ComplianceOperandType::Int:
        case ComplianceOperandType::date:
        {
            DWORD Int1 = 0;
            DWORD Int2 = 0;
            ComplianceOperand< int > Op1( exprType );
            ComplianceOperand< int > Op2( exprType );
            // Retrieve literal values from provider info, as required
            if ( provider1.length() )
            {
```

```
            propName1 = val1;
            if ( ! Providers.GetPropDWord( provider1, propName1, Int1 ) )
            {
               LOG_ALERT("CSCE: No DWord property named %s from provider %s\n", propName1.c_str(),
provider1.c_str() );
               break;  // error
            }
            Op1 = Int1;
         }
         else
         {
            Op1 = ComplianceOperand< int >( exprType, val1 );
         }
         if ( provider2.length() )
         {
            propName2 = val2;
            if ( ! Providers.GetPropDWord( provider2, propName2, Int2 ) )
            {
               LOG_ALERT("CSCE: No DWord property named %s from provider %s\n", propName2.c_str(),
provider2.c_str() );
               break;   // error
            }
            Op1 = Int2;
         }
         else
         {
            Op2 = ComplianceOperand< int >( exprType, val2 );
         }
         Op1.GetStringVal( val1 );
         Op2.GetStringVal( val2 );
         if ( exprOp == youngerDays )
            Op2 = (int) MakeComparisonDate( exprOp, Op1.m_val, Op2.m_val );
         ResultValue = Op1.DoCompare( exprOp, Op2 );
         break;
      }
      case ComplianceOperandType::version:
      case ComplianceOperandType::string:
      {
         // Retrieve literal values from provider info, as required
         if ( provider1.length() )
         {
            propName1 = val1;
            val1.clear();
            if (! Providers.GetPropString( provider1, propName1, val1 ) )
            {
               LOG_ALERT("CSCE: No string property named %s from provider %s\n", propName1.c_str(),
provider1.c_str() );
               break;  // error
            }
         }
```

```
        if ( provider2.length() )
        {
           propName2 = val2;
           val2.clear();
           if (! Providers.GetPropString( provider2, propName2, val2 ) )
           {
              LOG_ALERT("CSCE: No string property named %s from provider %s\n", propName2.c_str(),
provider2.c_str() );
              break;  // error
           }
        }
        ComplianceOperand< faststring > Op1( exprType, val1 );
        ComplianceOperand< faststring > Op2( exprType, val2 );
        Op1.GetStringVal( val1 );
        Op2.GetStringVal( val2 );
        ResultValue = Op1.DoCompare( exprOp, Op2 );
        break;
     }
     case ComplianceOperandType::expression:
     case ComplianceOperandType::err:
     default:
        LOG_ALERT("CSCE: Bad operand type in expression:\n%s\n", Expression.c_str() );
        return eErr;
  }
  // Record failure condition in the provider note
  if ( provider1.length() )
     ProviderReports.AddProviderPropValue( provider1, propName1.c_str(), val1, ResultValue );
  if ( provider2.length() )
     ProviderReports.AddProviderPropValue( provider2, propName2.c_str(), val2, ResultValue );
  return ResultValue;
}
ComplianceExpression::expResult ComplianceExpression::AggregateSubExpressions(
ComplianceOperator::Operator exprOp,
                                                const xmlstring          &Expression,
                                                ProviderReporterList     &Providers,
                                                ProviderReportList       &ProviderReports )
{
  DWORD dwBegin = -1;
  DWORD dwEnd = 10;
  expResult Result = eErr;
  if ( ! ( ( exprOp == ComplianceOperator::and ) ||
        ( exprOp == ComplianceOperator::or ) ) )
     return eErr;
  xmlstring subExpression;    // Apply the operator to all subexpressions
  while ( dwEnd != -1 )
  {
     Expression.GetXMLTag( "<expression>", subExpression, dwEnd, &dwBegin, &dwEnd );
     if ( subExpression.empty() )
        return Result;
     expResult subResult = EvaluateSubExpr( subExpression, Providers, ProviderReports );
```

```
      if ( Result == eErr )   // First time through -- assign and continue
      {
         Result = subResult;
         continue;
      }
      if ( subResult == eFalse )
      {
         if ( exprOp == ComplianceOperator::and )
            return eFalse;
      }
      else if ( subResult == eErr )
         return eErr;
      else    // eTrue
      {
         if ( exprOp == ComplianceOperator::or )
            return eTrue;
      }
   }
   return Result;
}
/*
 * If the expression is a collection of subexpressions, aggregate them.
 * Otherwise, compare the two operands.
 */
ComplianceExpression::expResult ComplianceExpression::EvaluateSubExpr( const xmlstring      &Expression,
                                                                       ProviderReporterList &Providers,
                                                                       ProviderReportList   &ProviderReports )
{
   DWORD dwBegin = -1;
   DWORD dwEnd = 10;
   ComplianceExpression::expResult Result = eErr;
   ComplianceOperandType::Type exprType = ComplianceOperandType::TypeFromString(
Expression.GetParameter( "type" ) );
   ComplianceOperator::Operator exprOp  = ComplianceOperator::OperatorFromString( Expression.GetParameter(
"operator" ) );
   fastistring unaryOp( Expression.GetParameter( "unaryOp" ) );     // TODO
   BOOL bNot = ( unaryOp == "not" );
   ProviderReportList CurrentReport;
   if ( exprType == ComplianceOperandType::expression )
   {
      Result = AggregateSubExpressions( exprOp, Expression, Providers, CurrentReport );
   }
   else
   {
      Result = CalcAndCompareOperands( exprType, exprOp, Expression, Providers, CurrentReport );
   }
   if ( bNot )
   {
      Result = ComplianceExpression::NegateResult( Result );
   }
```

```
    CurrentReport.SetListCompliant( Result );
    ProviderReports.MergeReports( CurrentReport );
    return Result;
}
//---------------------------------------------------------------------------
//---------------------------------------------------------------------------
BOOL CProviderReporter::GetPropDWord ( const faststring &Attribute, DWORD &dwResult ) const
{
    faststring value;
    if (! GetPropString( Attribute, value ) )
        return false;
    dwResult = atoi( value.c_str() );
    return true;
}
BOOL  CProviderReporter::GetPropString( const faststring &Attribute, faststring &fsResult ) const
{
    fsResult = m_status.GetParameter( Attribute );
    return ( fsResult.length() );
}
//---------------------------------------------------------------------------
//---------------------------------------------------------------------------
/*
 * Add the properties that haven't been referenced
 * yet.  For those that have, merge the result status.
 */
BOOL CProviderNote::AddPropValue( const faststring &Attribute,
                                  const faststring &Val,
                                  ComplianceExpression::expResult Result
                                )
{
    Property newProp( Attribute, Val, Result );
    std::pair< PropMap::iterator, bool > Insertion(
        m_properties.insert( PropMap::value_type( Attribute, newProp ) ) );
    if ( ! Insertion.second )   // Already found
    {
        Insertion.first->second.SetResult( Result );
        // Return false if no assignment
    }
    return TRUE;
}
BOOL CProviderNote::GetNoteXML( const faststring &fsProvider, xmlstring &Note ) const
{
    if ( ! m_properties.size() )
    {
        return FALSE;
    }
    Note += "       <provider ";
    Note.AddParameter( "name", fsProvider );
    // These fields are not used now: per-rule reporting.
//    Note.AddParameter( "status", ( m_Result == eTrue )?"1":"0" );
```

```
//    Note.AddParameter("compliance", m_compliance );
    Note += ">\n";
    PropMap::const_iterator it = m_properties.begin();
    for ( ; it != m_properties.end(); it++ )
    {
        if ( it->second.GetResult() != eTrue )
        {
            Note += "            <actualValue ";
            Note.AddParameter( "name", it->second.GetName() );
            Note.AddParameter( "val",  it->second.GetVal() );
            Note += "/>\n";
        }
    }
    Note += "        </provider>\n";
    return !! m_properties.size();
}
/*
 * For this provider, add any properties that haven't been referenced
 * yet.  For those that have, merge the result status.
 * Then, merge the new status of the provider itself.  First
 * Provider status is only set if the new status is "worse" than
 * the old, in order: true->false->error.
 */
BOOL CProviderNote::MergeNote( const CProviderNote ProviderNote )
{
    PropMap::const_iterator it = ProviderNote.m_properties.begin();
    for ( ; it != ProviderNote.m_properties.end(); it++ )
    {
        AddPropValue( it->second.GetName(),
                      it->second.GetVal(),
                      it->second.GetResult() );
        // Check if any assignments occur ??
    }
    // Merge provider-level result/compliance
    ComplianceResult::expResult newResult = MergeWorst( m_Result, ProviderNote.GetResult() );
    if ( m_Result == eTrue  && m_Result != newResult )
    {
        m_compliance = ProviderNote.GetCompliance();
        m_Result = newResult;
    }
    return TRUE;
}
void CProviderNote::ResetResults( ComplianceResult::expResult Result /*= eTrue*/ )
{
    PropMap::iterator it = m_properties.begin();
    for ( ; it != m_properties.end(); it++ )
    {
        it->second.SetResult( Result, TRUE );
    }
}
```

```
// This is for after a rule is evaluated.  If the note hasn't had a compliance
// assigned, we set the status and compliance here.
void CProviderNote::SetRuleResult( ComplianceResult::expResult Result,
                DWORD                           dwStatus,
                faststring                      fsCompliant )
{
   if ( ! m_bHasResults )
   {
      m_bHasResults = TRUE;
      if ( Result != eTrue )
      {
         m_Result = MergeWorst( m_Result, Result );
      }
      m_compliance = fsCompliant;
   }
}
//---------------------------------------------------------------------
//---------------------------------------------------------------------
void ProviderReporterList::AddProvider( const fastistring &fstrName, IProviderReporterRef *pProvider )
{
   ZComPtr< IProviderReporterRef > ProviderRef( pProvider );
   m_providers.insert( ProviderMap::value_type( fstrName, ProviderRef ) );
}
BOOL ProviderReporterList::GetPropDWord ( const fastistring &providerName, const fastistring &Attribute, DWORD
&dwResult ) const
{
   ProviderMap::const_iterator it = m_providers.find( providerName );
   if ( it != m_providers.end() )
      return it->second->GetPropDWord( Attribute, dwResult );
   return FALSE;
}
BOOL ProviderReporterList::GetPropString( const fastistring &providerName, const fastistring &Attribute, faststring
&fsResult ) const
{
   ProviderMap::const_iterator it = m_providers.find( providerName );
   if ( it != m_providers.end() )
      return it->second->GetPropString( Attribute, fsResult );
   return FALSE;
}
void ProviderReporterList::AddSystemProvider()
{
   AddProvider( "system", new CSystemProvider() );
}
/*
 * ---------------------------------------------------------------------
 * ---------------------------------------------------------------------
 */
BOOL ProviderReportList::GetProviderNote( const fastistring &fsProvider, xmlstring &xsNote ) const  // empty
fsProvider = ALL
{
```

```
    NoteMap::const_iterator it;
    if ( fsProvider.length() )
    {
        it = m_notes.find( fsProvider );
        return it->second.GetNoteXML( fsProvider, xsNote );
    }
    else
    {
        it = m_notes.begin();
        if ( it == m_notes.end() )
        {
            return FALSE;
        }
        for ( ; it != m_notes.end(); it++ )
        {
            it->second.GetNoteXML( it->first, xsNote );
        }
        return TRUE;
    }
}
/*
 * Either add the property to this provider's list, or merge the
 * result to the one already there.
 */
BOOL ProviderReportList::AddProviderPropValue( const fastistring &Provider,
                                               const fastistring &Attribute,
                                               const fastistring &ActualVal,
                                               ComplianceExpression::expResult Result
                                              )
{
    std::pair< NoteMap::iterator, bool > Insertion(
        m_notes.insert( NoteMap::value_type( Provider, CProviderNote() ) ) );
    return Insertion.first->second.AddPropValue( Attribute, ActualVal, Result );
}
/*
 * Either add the note to this report or merge the note we already have
 */
BOOL ProviderReportList::MergeProviderNote( const fastistring &fsProvider, const CProviderNote &Note )
{
    std::pair< NoteMap::iterator, bool > Insertion(
        m_notes.insert( NoteMap::value_type( fsProvider, CProviderNote() ) ) );
    return Insertion.first->second.MergeNote( Note );
}
/*
 * For each provider in the given report, merge the new results with ours
 */
BOOL ProviderReportList::MergeReports( const ProviderReportList &Reports )
{
    for ( NoteMap::const_iterator it = Reports.m_notes.begin(); it != Reports.m_notes.end(); it++ )
    {
```

```
      MergeProviderNote( it->first, it->second );
   }
   return TRUE;
}
/*
 * For each subexpression result, we need to walk the list of referenced
 * properties on the way back out and assign their culpability for the
 * topmost result.
 */
void ProviderReportList::SetListCompliant( ComplianceResult::expResult Result /*= eTrue*/ )
{
   for ( NoteMap::iterator it = m_notes.begin(); it != m_notes.end(); it++ )
   {
      it->second.ResetResults( Result );
   }
}
void ProviderReportList::SetRuleResult( ComplianceResult::expResult Result,
                                        DWORD                       dwRuleStatus,
                                        faststring                  fsCompliant )
{
   if ( Result != ComplianceResult::eTrue )
   {
      for ( NoteMap::iterator it = m_notes.begin(); it != m_notes.end(); it++ )
      {
         it->second.SetRuleResult( Result, dwRuleStatus, fsCompliant );
      }
   }
}
/*
 * Return format for compliance info: strings terminated with 0
 *   Item              Type       Size
 *   ------------------------------------------------
 *   TotalCount        DWORD      sizeof( DWORD )     Number of items
 *
 *   dwCompliance      DWORD      sizeof( DWORD )     +
 *   szConnectionId    string     variable            |
 *   szSessionId       string     variable            +-- First Item
 *   szCompliance      string     variable            |
 *   szReport          string     variable            +
 *
 *   dwCompliance      DWORD      sizeof( DWORD )     +
 *   szConnectionId    string     variable            |
 *   szSessionId       string     variable            +-- Second Item
 *   szCompliance      string     variable            |
 *   szReport          string     variable            +
 *                                      ...etc.
 */
int ComplianceReport::DumpToBuffer( char *pBuf, int size )
{
   int newSize =   sizeof( DWORD ) +
```

```
                m_szConnectionId.length() + 1 +
                m_szSessionId.length()    + 1 +
                m_szCompliance.length()   + 1 +
                m_xsReport.length()       + 1;
    if ( size < newSize )
        return FALSE;
    DWORD dwComplianceVal = m_dwStatus;
    // No news is good news.
    if ( m_dwStatus == COMPLY_STATUS_NULL )
        dwComplianceVal = COMPLY_STATUS_SUCCESS;
    *( ( DWORD *) pBuf ) = dwComplianceVal;
    pBuf += sizeof( DWORD );
    strcpy( pBuf, m_szConnectionId );
    pBuf += m_szConnectionId.length() + 1;
    strcpy( pBuf, m_szSessionId );
    pBuf += m_szSessionId.length() + 1;
    strcpy( pBuf, m_szCompliance );
    pBuf += m_szCompliance.length() + 1;
    strcpy( pBuf, m_xsReport );
    pBuf += m_xsReport.length() + 1;
    *pBuf = '\0';
    return newSize;
}
void ComplianceReport::MergeRuleResult( DWORD dwResult,
                                const ComplianceRule &EvaluatedRule,
                                const ProviderReportList &ProviderReports )
{
    MergeComplianceResult( dwResult, EvaluatedRule.GetCompliance() );
    if ( EvaluatedRule.IsDirty() )
    {
        m_bDirtyRules = TRUE;
        EvaluatedRule.GetRuleReport( ProviderReports, m_xsReport );
    }
}
void ComplianceReport::MergeComplianceResult( DWORD dwResult, const faststring &fsCompliance )
{
    // TODO: merge some text
    // TODO: make sure we wanted text...
    if ( MergeComplianceResult( dwResult ) )
    {
        m_szCompliance = fsCompliance;
    }
}
// Return of TRUE indicates that a failed result was merged in.
// Note that a failure code always overrides the merged result.
BOOL ComplianceReport::MergeComplianceResult( DWORD dwResult )
{
    switch ( m_dwStatus )
    {
        case COMPLY_STATUS_ERROR:
```

```
            break;      // Do not overwrite
        case COMPLY_STATUS_FAILURE:
            if ( dwResult != COMPLY_STATUS_ERROR )
                break;
        case COMPLY_STATUS_REQUIRES_UPDATE:
            if ( ( dwResult != COMPLY_STATUS_FAILURE ) &&
                 ( dwResult != COMPLY_STATUS_ERROR ) )
                break;
            // else fall through
        case COMPLY_STATUS_NULL:
        case COMPLY_STATUS_SUCCESS:
        {
            m_dwStatus = dwResult;
            return ( ( dwResult == COMPLY_STATUS_REQUIRES_UPDATE ) ||
                    ( dwResult == COMPLY_STATUS_FAILURE ) );
            break;
        }
    }
    return FALSE;
}
BOOL ComplianceReport::AddProviderReports( const xmlstring &xsProviderReports )
{
    m_xsReport += xsProviderReports;
    return !! m_xsReport.length();
}
void ComplianceReport::GetReport( xmlstring &xsReport )
{
//    char buf[40] = "\0";
//    xsReport = "<complianceReport ";
//    xsReport.AddParameter( "status", _itoa( m_dwStatus, buf, 10 ) );    // allow negative value
//    xsReport.AddParameter("compliance", m_compliance );
//    xsReport += " />\n"
    xsReport += m_xsReport;
//    xsReport += "</complianceReport>\n";
}
// For doing a heartbeat compliance check after the main evaluation is done,
// but of course could be used with any rule.
void ISComplianceReport::ReEvaluateRule( ImpliedComplianceRule &CompRule )
{
    ProviderReporterList Providers;     // dummy for passing reference
    ProviderReportList   Reports;       // dummy for passing reference
    DWORD dwResult = CompRule.Evaluate( Providers, Reports );
    Providers.MergeReports( Reports );
    MergeRuleResult( dwResult, CompRule, Reports );
}
void ISComplianceReport::CalcHBCompliance( void )
{
    if ( IsCompliant() )
        m_dwNoncompHB = 0;
    else
```

```
        m_dwNoncompHB++;
// SSNOTE: TODO: do we do this?
//    if ( m_dwNoncompHB >= 10 )
//    {
//        MergeComplianceResult( COMPLY_STATUS_FAILURE, COMPLYSTR_HB_NONCOMPLIANT );
//    }
}
DWORD ComplianceRule::Evaluate( ProviderReporterList &Providers, ProviderReportList &Reports )
{
    DWORD dwNewResult = COMPLY_STATUS_SUCCESS;
    ComplianceResult::expResult Result = m_expression.Evaluate( Providers, Reports );
    Reports.SetRuleResult( Result, m_dwRuleStatus, m_Compliant );
    if ( Result == ComplianceResult::eTrue )
    {
        dwNewResult = COMPLY_STATUS_SUCCESS;
    }
    else if ( Result == ComplianceResult::eFalse )
    {
        dwNewResult = COMPLY_STATUS_REQUIRES_UPDATE;
    }
    else
    {
        dwNewResult = COMPLY_STATUS_FAILURE;
    }
    SetNewResult( dwNewResult );
    // If this is an observe-only rule, always return "compliant"
    // though we maintain a failed status.
    if ( m_dwRuleStatus == 1 )
    {
        dwNewResult = COMPLY_STATUS_SUCCESS;
    }
    return dwNewResult;
}
void ComplianceRule::GetRuleReport( const ProviderReportList &ProviderReports,
                                    xmlstring &xsReport ) const
{
    xsReport += "    <complianceRule ";
    if ( m_Name.length() )
    {
        xsReport.AddParameter( "name", m_Name );
    }
    char buf[40] = "\0";
    DWORD dwOutputStatus = 1;
    if ( ! m_dwRuleStatus )
    {
        dwOutputStatus = GetLastResult();
    }
    xsReport.AddParameter( "status", _itoa( dwOutputStatus, buf, 10 ) );    // allow negative value
    xsReport.AddParameter("compliance", m_Compliant );
    xsReport += ">\n";
```

```
    ProviderReports.GetProviderNote( fastistring(""), xsReport );
    xsReport += "    </complianceRule>\n";
}
DWORD ImpliedComplianceRule::Evaluate( ProviderReporterList &Providers, ProviderReportList &Reports )
{
    DWORD dwReturn = COMPLY_STATUS_SUCCESS;
    ComplianceResult::expResult Result = ComplianceResult::eFalse;
    if ( ! m_bAuthorized )
    {
        m_Compliant = COMPLYSTR_ZL_CONNECTION_STATUS;
        dwReturn = COMPLY_STATUS_REQUIRES_UPDATE;
    }
    else if ( m_dwMissedHeartbeats >= m_dwHeartbeatsRestrict  )
    {
        if ( m_dwMissedHeartbeats >= m_dwHeartbeatsTerminate )
        {
            m_Compliant = COMPLYSTR_HB_MISSED_SEVERE;
            dwReturn = COMPLY_STATUS_FAILURE;
        }
        else
        {
            m_Compliant = COMPLYSTR_HB_MISSED_WARNING;
            dwReturn = COMPLY_STATUS_REQUIRES_UPDATE;
        }
        char szBuf[40];
        Reports.AddProviderPropValue( "ZoneLabs",
                                      "missedHeartbeats",
                                      _itoa( m_dwMissedHeartbeats, szBuf, 10 ),
                                      ComplianceExpression::eFalse );
    }
    else
    {
        Result = ComplianceResult::eTrue;
    }
    // SSNOTE: todo: add the properties to the report list
    Reports.SetRuleResult( Result, m_dwRuleStatus, m_Compliant );
    SetNewResult( dwReturn );
    return dwReturn;
}
/*
 * Record the new result.  If the result is different than last
 * time, notify any registered listener.
 */
void ComplianceRule::SetNewResult( DWORD dwNewResult )
{
    DWORD dwLastResult = m_dwLastResult;
    m_dwLastResult = dwNewResult;
    m_bDirty = ( dwNewResult != dwLastResult );
    if ( m_bDirty && m_pListener )
    {
```

```
      m_pListener->NotifyStatusChange( *this );
   }
}
BOOL ComplianceChecker::CalculateCompliance(  ProviderReporterList &Providers,
                                    ComplianceReport &Report )
{
   Report.clear();
   while ( ComplianceRule *pRule = PopFrontRule() )
   {
      ProviderReportList ProviderReports;     // Individual rule result/report
      DWORD dwResult = pRule->Evaluate( Providers, ProviderReports );
      Report.MergeRuleResult( dwResult, *pRule, ProviderReports );    // Merge high level compliance result
      Providers.MergeReports( ProviderReports );      // Merge provider property lists - though unused now
      pRule->Release();
   }
   // Per-provider reports no longer calculated.
   return TRUE;
}
// router.c
// Copyright (c) 2003. Zone Labs, Inc. All Rights Reserved.
/*
ROUTER.C - CMP Challenge/Response
*/
#include <windows.h>
#include <winsock.h>
#include <wsnwlink.h>
#include "rsaapi.h"
#include "vscmp.h"
#include "rtcmp.h"
#include "vsutils.h"
#include <stdlib.h>
#include <stdio.h>
#include <memory.h>
#include <string.h>
// Broadcast addresses
#define SUBNET_BROADCAST_ADDR(addr, mask) (((addr) & (mask)) | (~(mask)))
#define SUBNET_MASK 0x00FFFFFF // network order
/*****************************************************************/
/**              Sample Client Status Table                     **/
/** if current router control table has IP address columnn  **/
/**     please add one DWORD, whose 0-30 bits for client status **/
/**     and 31 bit for license status                          **/
/**     CMPR_LICENSE_ACCEPTED=0x80000000                       **/
/*****************************************************************/
typedef struct {
   DWORD  dwIPAddr;              // client IP address
   DWORD  dwStatus;              // client status
//    UCHAR  cLicFlag;             // client license status
} CLIENT_STATUS, *PCLIENT_STATUS;
/*****************************************************************/
```

```
/**  All the tGlobalClient related code should be rewriten  **/
/**     if the table format is different !!!!!!!!!!!!!!!!!!!!  **/
/*****************************************************************/
// can adjust to the real router control table size
#define  MAX_TABLEMEMBER    256
CLIENT_STATUS tGlobalClient[MAX_TABLEMEMBER];
// if ctrl-c/ctrl_break
WORD wLoopControl=1;
// WSA flag
BOOL fStarted = FALSE;
BOOL ParseVersion(CHAR *pcVersStr, PCMPVERS pVers)
{
 if (pcVersStr && (*pcVersStr))
 {
  INT i = 0;
  CHAR *pStr = pcVersStr;
  CHAR *pDot = strchr(pStr, '.');
  while (pDot)
  {
   *pDot = 0;
          if(*pStr>'9' || *pStr<'0')
           return FALSE;
   pVers->wVers[i] = (WORD) atoi(pStr);
   *pDot = '.';
   pStr = pDot + 1;
   if (++i > 3)
           return FALSE;
   pDot = strchr(pStr, '.');
  }
  if (i==3 && pStr && *pStr<='9' && *pStr>='0')
      {
   pVers->wVers[i] = (WORD) atoi(pStr);
          return TRUE;
      }
 }
 return FALSE;
}
DWORD StringToAddress(CHAR *pStrAddr)
{
    DWORD dwResult=0;
    DWORD dwTempResult=0;
 if (pStrAddr && (*pStrAddr))
 {
  INT i = 0;
  CHAR *pStr = pStrAddr;
  CHAR *pDot = strchr(pStr, '.');
  while (pDot)
  {
   *pDot = 0;
          if(*pStr>'9' || *pStr<'0')
```

```
            return 0;
   if( (dwTempResult=(WORD)atoi(pStr))>255 )
            return 0;
           dwResult|=dwTempResult<<((3-i)*8);
   *pDot = '.';
   pStr = pDot + 1;
   if (++i > 3)
            return 0;
   pDot = strchr(pStr, '.');
  }
  if (i==3 && pStr && *pStr<='9' && *pStr>='0')
       {
          if( (dwTempResult=(WORD)atoi(pStr))<=255 )
          {
             dwResult|=dwTempResult;
             return dwResult;
          }
       }
 }
 return 0;
}
//   real sample
//   need to be rewriten if the client table is different !!!!!!!!!!
void PrintClientStatus(char* pWhere)
{
    int i, j=0;
 DWORD dwIPAddress,dwTempStatus;
 char sShow[60];
 char* sMessage[3]={"Client Exempt !","Client Timeout !","No Client response !"};
 char* sMessage2[7]={"Wrong Client Version !","Invalid License !","No more license !","Antivirus not running !","Wrong
antivirus version !","Antivirus not autoupdate !","Antivirus not realtime Monitor !"};
    printf("\n");
    printf("%s\n",pWhere);
    for(i=0; i<MAX_TABLEMEMBER; i++)
    {
  dwIPAddress=SwapDWord(tGlobalClient[i].dwIPAddr);
       dwTempStatus=tGlobalClient[i].dwStatus & CMPR_MASK;
  if(dwTempStatus==CMPR_CLIENT_EXEMPT)
   strcpy(sShow,sMessage[0]);
  else if(dwTempStatus>256)
   sprintf(sShow,"Time Stamp=%d",dwTempStatus-256);
  else if(dwTempStatus>39)
   sprintf(sShow,"Status=%d",dwTempStatus);
  else if(dwTempStatus>32)
   strcpy(sShow,sMessage2[dwTempStatus-33]);
  else if(dwTempStatus>2)
   sprintf(sShow,"Status=%d",dwTempStatus);
  else if(dwTempStatus>0) // shouldn't happen
   strcpy(sShow,sMessage[dwTempStatus]);
  else
```

```
  continue;
      printf("<IP address=%d.%d.%d.%d>  <License status=%d>  <%s>\n",
       (dwIPAddress & 0xff000000)>>24,(dwIPAddress & 0x00ff0000)>>16,
       (dwIPAddress & 0x0000ff00)>>8, dwIPAddress & 0x000000ff,
       tGlobalClient[i].dwStatus & CMPR_LICENSE_ACCEPTED ? 1 : 0,sShow);
 j++;
   }
 if(!j)
      printf("No client information !\n");
}
//   real sample
//   need to be rewriten if the client table is different !!!!!!!!!!
void InitializeGlobals(DWORD dwIPAddress)
{
   int i;
// CLIENT_STATUS initStatus={0,0,0};
 CLIENT_STATUS initStatus={0,0};
 dwIPAddress=dwIPAddress & SwapDWord(SUBNET_MASK);
 // assume IPAddr starts from xxx.xxx .0.0
   for(i=0; i<MAX_TABLEMEMBER; i++)
   {
      memcpy(&tGlobalClient[i],&initStatus,sizeof(CLIENT_STATUS));
 tGlobalClient[i].dwIPAddr=SwapDWord(dwIPAddress);
      if(((i%100)==8) || (i==111) /*|| (i==33)*/)
         tGlobalClient[i].dwStatus=CMPR_CLIENT_EXEMPT;
 dwIPAddress++;
   }
 PrintClientStatus("======Initialize======");
}
/* validate license key, 27 characters
*/
BOOL CMPValidateLicenseKey(char* InputKey)
{
   int i;
   if(strlen(InputKey) != 27)
      return FALSE;
   for(i=0; i<27; i++)
   {
      if(InputKey[i]<'0' ||
        (InputKey[i]>'9' && InputKey[i]<'A') ||
        (InputKey[i]>'Z' && InputKey[i]<'a') ||
         InputKey[i]>'z')
          return FALSE;
   }
   return TRUE;
}
#if DO_USERPROMPT
/*
return: DWORD, packet size
1. buffers
```

```
 UCHAR* packetChallenge
 UCHAR* packetEncrypted
2. challenge arguments
 DWORD dwRouteIPAddress
 UCHAR* pcmpRouteVersion
 DWORD dwRouteSessionID
 DWORD dwResponseTime
3. user prompt option argument
 char* pUserPrompt
*/
DWORD PrepareUserPromptPacket(UCHAR* packetChallenge, UCHAR* packetEncrypted,
                    DWORD dwRouteIPAddress, UCHAR* pcmpRouteVersion,
                    DWORD dwRouteSessionID, DWORD dwResponseTime, char* pUserPrompt)
{
   PRT_CHALLENGE_PACKET pTempChallenge;
   WORD wTempPacketSize,wTempOptionStart;
   DWORD dwPresentTime;
   BOOL rStatus;
   // create non heartbeat challenge packet
   dwPresentTime=GetTickCount()/1000;
   rStatus=CMPCreateChallengePacket(packetChallenge, CMP_MAX_CHALLENGE, CMPP_LINKSYS,
                        dwRouteIPAddress, pcmpRouteVersion, dwRouteSessionID, dwResponseTime, dwPresentTime);
   pTempChallenge=(PRT_CHALLENGE_PACKET)packetChallenge;
   wTempOptionStart=pTempChallenge->wPacketSize;
   if(rStatus)
      rStatus=CMPAddUserPromptOption(packetChallenge, CMP_MAX_CHALLENGE, pUserPrompt);
   if(rStatus)
   {
    wTempPacketSize=pTempChallenge->wPacketSize;
      // swap
      SwapRouterChallenge((PRT_CHALLENGE_PACKET)packetChallenge);
      SwapOptUserPrompt((POPT_USER_PROMPT)((UCHAR*)packetChallenge+wTempOptionStart));
      // checksum
      pTempChallenge->dPacketCRC=CheckSum((WORD*)packetChallenge, wTempPacketSize, 0);
      // ecrypt the packet
      return CMPEncryptPacket(packetChallenge, wTempPacketSize, packetEncrypted, CMP_MAX_CHALLENGE);
   }
   return 0;
}
#endif
/*
return: DWORD, packet size
1. buffers
 UCHAR* packetChallenge
 UCHAR* packetEncrypted
2. challenge arguments
 DWORD dwRouteIPAddress
 UCHAR* pcmpRouteVersion
 DWORD dwRouteSessionID
 DWORD dwResponseTime
```

```
3. license key option arguments
 WORD wNumberOfUsers
 char* pLicenseKey
*/
DWORD PrepareLicenseKeyPacket(UCHAR* packetChallenge, UCHAR* packetEncrypted,
                DWORD dwRouteIPAddress, UCHAR* pcmpRouteVersion, DWORD dwRouteSessionID,
                DWORD dwResponseTime,WORD wNumberOfUsers, char* pLicenseKey)
{
   PRT_CHALLENGE_PACKET pTempChallenge;
   WORD wTempPacketSize,wTempOptionStart;
   DWORD dwPresentTime;
   BOOL rStatus;
   // create non heartbeat challenge packet
   dwPresentTime=GetTickCount()/1000;
   rStatus=CMPCreateChallengePacket(packetChallenge, CMP_MAX_CHALLENGE, CMPP_LINKSYS,
                dwRouteIPAddress, pcmpRouteVersion, dwRouteSessionID, dwResponseTime, dwPresentTime);
   pTempChallenge=(PRT_CHALLENGE_PACKET)packetChallenge;
   wTempOptionStart=pTempChallenge->wPacketSize;
   if(rStatus)
      rStatus=CMPAddLicenseOption(packetChallenge, CMP_MAX_CHALLENGE, wNumberOfUsers, pLicenseKey);
   if(rStatus)
   {
    wTempPacketSize=pTempChallenge->wPacketSize;
      // swap
      SwapRouterChallenge((PRT_CHALLENGE_PACKET)packetChallenge);
      SwapOptLicense((POPT_ZL_LICENSE)((UCHAR*)packetChallenge+wTempOptionStart));
      // checksum
      pTempChallenge->dPacketCRC=CheckSum((WORD*)packetChallenge, wTempPacketSize, 0);
      // ecrypt the packet
      return CMPEncryptPacket(packetChallenge, wTempPacketSize, packetEncrypted, CMP_MAX_CHALLENGE);
   }
   return 0;
}
/*
return: DWORD, packet size
1. buffers
 UCHAR* packetChallenge
 UCHAR* packetEncrypted
2. challenge arguments
 DWORD dwRouteIPAddress
 UCHAR* pcmpRouteVersion
 DWORD dwRouteSessionID
 DWORD dwResponseTime
3. ZAP version option argument
 UCHAR* cmpMinimumVersion
*/
DWORD PrepareVersionInquiryPacket(UCHAR* packetChallenge, UCHAR* packetEncrypted,
                DWORD dwRouteIPAddress, UCHAR* pcmpRouteVersion, DWORD dwRouteSessionID,
                DWORD dwResponseTime,UCHAR* cmpMinimumVersion)
{
```

```
    PRT_CHALLENGE_PACKET pTempChallenge;
    WORD wTempPacketSize,wTempOptionStart;
    DWORD dwPresentTime;
    BOOL rStatus;
    // create non heartbeat challenge packet
    dwPresentTime=GetTickCount()/1000;
    rStatus=CMPCreateChallengePacket(packetChallenge, CMP_MAX_CHALLENGE, CMPP_LINKSYS,
                        dwRouteIPAddress, pcmpRouteVersion, dwRouteSessionID, dwResponseTime, dwPresentTime);
    pTempChallenge=(PRT_CHALLENGE_PACKET)packetChallenge;
    wTempOptionStart=pTempChallenge->wPacketSize;
    if(rStatus)
       rStatus=CMPAddClientVersionOption(packetChallenge, CMP_MAX_CHALLENGE, CMPP_ZAPRO,
cmpMinimumVersion);
    if(rStatus)
    {
     wTempPacketSize=pTempChallenge->wPacketSize;
       // swap
       SwapRouterChallenge((PRT_CHALLENGE_PACKET)packetChallenge);
       SwapOptClientVersion((POPT_CLIENT_VERSION)((UCHAR*)packetChallenge+wTempOptionStart));
       // checksum
       pTempChallenge->dPacketCRC=CheckSum((WORD*)packetChallenge, wTempPacketSize, 0);
       // ecrypt the packet
       return CMPEncryptPacket(packetChallenge, wTempPacketSize, packetEncrypted, CMP_MAX_CHALLENGE);
    }
    return 0;
}
#if DO_ANTIVIRUS
/*
return: DWORD, packet size
1. buffers
 UCHAR* packetChallenge
 UCHAR* packetEncrypted
2. Challenge arguments
 DWORD dwRouteIPAddress
 UCHAR* pcmpRouteVersion
 DWORD dwRouteSessionID
 DWORD dwResponseTime
3. AV option arguments
 DWORD dwAVProductID
 UCHAR* pcmpAVVersion
 BOOL bEnforceAutoUpdate
 BOOL bEnforceRealTimeMonitor
*/
DWORD PrepareAVInquiryPacket(UCHAR* packetChallenge, UCHAR* packetEncrypted,
                    DWORD dwRouteIPAddress, UCHAR* pcmpRouteVersion, DWORD dwRouteSessionID,
                    DWORD dwResponseTime, DWORD dwAVProductID, UCHAR* pcmpAVVersion,
      BOOL bEnforceAutoUpdate, BOOL bEnforceRealTimeMonitor)
{
    PRT_CHALLENGE_PACKET pTempChallenge;
    WORD wTempPacketSize,wTempOptionStart;
```

```
    DWORD dwPresentTime;
    BOOL rStatus;
    // create non heartbeat challenge packet
    dwPresentTime=GetTickCount()/1000;
    rStatus=CMPCreateChallengePacket(packetChallenge, CMP_MAX_CHALLENGE, CMPP_LINKSYS,
                        dwRouteIPAddress, pcmpRouteVersion, dwRouteSessionID, dwResponseTime, dwPresentTime);
    pTempChallenge=(PRT_CHALLENGE_PACKET)packetChallenge;
    wTempOptionStart=pTempChallenge->wPacketSize;
    if(rStatus)
       rStatus=CMPAddAntivirusOption(packetChallenge, CMP_MAX_CHALLENGE, dwAVProductID, pcmpAVVersion,
                   bEnforceAutoUpdate, bEnforceRealTimeMonitor, 0, 0);
    if(rStatus)
    {
     wTempPacketSize=pTempChallenge->wPacketSize;
       // swap
       SwapRouterChallenge((PRT_CHALLENGE_PACKET)packetChallenge);
       SwapOptAntiVirus((POPT_ANTI_VIRUS)((UCHAR*)packetChallenge+wTempOptionStart));
       // checksum
       pTempChallenge->dPacketCRC=CheckSum((WORD*)packetChallenge, wTempPacketSize, 0);
       // ecrypt the packet
       return CMPEncryptPacket(packetChallenge, wTempPacketSize, packetEncrypted, CMP_MAX_CHALLENGE);
    }
    return 0;
}
#endif
/*
return: DWORD, packet size
1. buffers
 UCHAR* packetChallenge
 UCHAR* packetEncrypted
2. challenge arguments
 DWORD dwRouteIPAddress
 UCHAR* pcmpRouteVersion
 DWORD dwRouteSessionID
 DWORD dwResponseTime
*/
DWORD PrepareHeartBeatPacket(UCHAR* packetChallenge, UCHAR* packetEncrypted,
     DWORD dwRouteIPAddress, UCHAR* pcmpRouteVersion, DWORD dwRouteSessionID, DWORD
dwResponseTime)
{
    PRT_CHALLENGE_PACKET pTempChallenge;
    WORD wTempPacketSize;
    DWORD dwPresentTime;
    BOOL rStatus;
    // create non heartbeat challenge packet
    dwPresentTime=GetTickCount()/1000;
    rStatus=CMPCreateChallengePacket(packetChallenge, CMP_MAX_CHALLENGE, CMPP_LINKSYS,
                        dwRouteIPAddress, pcmpRouteVersion, dwRouteSessionID, dwResponseTime, dwPresentTime);
    if(rStatus)
    {
```

```
 pTempChallenge=(PRT_CHALLENGE_PACKET)packetChallenge;
 wTempPacketSize=pTempChallenge->wPacketSize;
      SwapRouterChallenge((PRT_CHALLENGE_PACKET)packetChallenge);
      // checksum
      pTempChallenge->dPacketCRC=CheckSum((WORD*)packetChallenge, wTempPacketSize, 0);
      // ecrypt the packet
      return CMPEncryptPacket(packetChallenge, wTempPacketSize, packetEncrypted, CMP_MAX_CHALLENGE);
   }
   return 0;
}
/*
return: int, socket status
Windows special
*/
int CMPGetSocket(SOCKET *pSock, DWORD dwIPAddr, INT iTimeout)
{
 INT iStatus = SOCKET_ERROR;
 SOCKET sock = INVALID_SOCKET;
 WSADATA wsaData;
 int iRetVal;
 iRetVal = WSAStartup ( MAKEWORD ( 1,1 ), &wsaData );
 if (iRetVal != 0)
 {
  printf( "/n WSAStartup=%d", iRetVal );
  return iStatus;
 }
   fStarted = TRUE;
 sock = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
 if (sock == INVALID_SOCKET)
  iStatus = SOCKET_ERROR;
 else if (dwIPAddr)
 {
  struct sockaddr_in addr;
  memset(&addr, 0, sizeof(addr));
  addr.sin_family = AF_INET;
  addr.sin_port = SwapWord((WORD)CMP_PORT);
  addr.sin_addr.s_addr = dwIPAddr;
  iStatus = bind(sock, (const struct sockaddr *) &addr, sizeof(addr));
  if (iStatus)
  {
   printf( "\n bind=%d", WSAGetLastError ( ) );
   closesocket(sock);
   sock = INVALID_SOCKET;
  }
 }
 // do we ever want to set a send timeout?
 if (iTimeout && !iStatus)
  iStatus=setsockopt(sock, SOL_SOCKET, SO_RCVTIMEO, (char *) &iTimeout, sizeof(iTimeout));
 if (iStatus)
 {
```

```
  printf( "\n setsockopt=%d", WSAGetLastError ( ) );
 }
 *pSock = sock;
 return iStatus;
}
/*
return: DWORD, length received from the client
Windows special
*/
DWORD CMPListenToClient(SOCKET rsock, DWORD dwServerAddr,
                        UCHAR* cBufReceived, DWORD dwBufLen, DWORD* dwClientIP)
{
 INT iLen=0;
 int iSize;
 struct sockaddr_in remAddr;
 memset(cBufReceived, 0, dwBufLen);
 memset(&remAddr, 0, sizeof(remAddr));
 iSize = sizeof(remAddr);
 if(rsock!=INVALID_SOCKET)
  iLen = recvfrom(rsock, cBufReceived, dwBufLen, 0, (struct sockaddr *) &remAddr, &iSize);
 if (iLen > 0)
    {
  if (((remAddr.sin_addr.s_addr & SUBNET_MASK) == (dwServerAddr & SUBNET_MASK)) &&
   ((remAddr.sin_port == SwapWord((WORD)CMP_PORT))))
       {
          *dwClientIP=remAddr.sin_addr.s_addr;
          return iLen;
       }
    }
 return 0;
}
/* close socket
Windows special
*/
void CMPKillSocket(SOCKET rSock)
{
 if (rSock != INVALID_SOCKET)
  closesocket(rSock);
    if ( TRUE == fStarted )
    {
  WSACleanup ( );
    }
 fStarted=FALSE;
}
//
// Broadcasts challenge packet
// Windows special
void CMPBroadcast(SOCKET rSock, UCHAR* packetChallenge, DWORD dwPkLen, DWORD dwServerAddr)
{
 int iSendOut;
```

```
 struct sockaddr_in remAddr;
 memset(&remAddr, 0, sizeof(remAddr));
 remAddr.sin_family = AF_INET;
 remAddr.sin_port = SwapWord((WORD)CMP_PORT);
 remAddr.sin_addr.s_addr = SUBNET_BROADCAST_ADDR(dwServerAddr, SUBNET_MASK);
 if(rSock!=INVALID_SOCKET)
  iSendOut=sendto(rSock, (const char *)packetChallenge, dwPkLen, 0, (const struct sockaddr *) &remAddr,
sizeof(remAddr));
}
/* Count the license number being used from the sample client table
   need to be rewriten if the client table is different !!!!!!!!!!
   DWORD dwTimeout: 4 times of client response time
   !!!!!!!!!!!!!!!may need to be rewriten!!!!!!!!!!!!!!!!!!!!
*/
WORD CMPCountLicenses(DWORD dwTimeout)
{
    int i;
    DWORD dwPresentTime;
    DWORD dwTempStatus;
    WORD wTempNumber=0;
    dwPresentTime=GetTickCount()/1000;
    for(i=0; i<MAX_TABLEMEMBER; i++)
    {
        dwTempStatus=tGlobalClient[i].dwStatus & CMPR_MASK;
        if((tGlobalClient[i].dwStatus & CMPR_LICENSE_ACCEPTED) &&
            dwTempStatus > 256 &&
            (dwPresentTime-dwTempStatus-256) < dwTimeout)
        wTempNumber++;
    }
    return wTempNumber;
}
/* for the purpose to catch ctrl-c & ctrl-break */
// PC special
BOOL CtrlHandler(DWORD dwEvent)
{
    if(dwEvent==CTRL_C_EVENT || dwEvent==CTRL_BREAK_EVENT)
      wLoopControl=0;
    return FALSE;
}
/*
To test heartbeat
 router /c
 router /s
 router /p prompt string
       example: router /p ZAP is enforced to run
 router /v version number
       example router /v 2.6.0.300
 router /l license string
       example: router /l abcdefghijklmnopqrstuvwxyz1
 router /a
```

```
        example
To test authorizing traffic(not being tested yet)
 router /t IP address
        example: router /t 192.66.55.218
to talk to cliens and update the client table, rtest /c
to do control panel setting, talk to cliens and update the client table, rtest /s
to talk to cliens with a prompt and update the client table, rtest /p prompt string
to talk to cliens with a version inquiry and update the client table, rtest /v last 3 digit of version number
to talk to cliens with a license inquiry and update the client table, rtest /l license string
to talk to cliens with a Antivirus inquiry and update the client table, rtest /a
to authorize traffic, rtest /t IP address
*/
void main(int argc,char** argv)
{
    BOOL bSettings=FALSE;
    //input from the control panel
    //ZAP related
    BOOL bZAPEnabled=TRUE;
    DWORD gdwLinkSysZoneLabsWeb=0;
    DWORD gdwResponseTime=3;// client repsonce time
    char* newUserPrompt="LinkSys security enforcement";//
    CMPVERS gcmpZAPVersion;
//          "5-5-5-6-6" "AQ13E-B2SGE-93JSE-HGXY36-YS6TT8";//27 CHARACTERS ?
    char* newLicenseKey="jsgn7rffsigik024sub23drjp00";//27 CHARACTERS ?
 char cUserLicense[28];
    char cUserPrompt[60];
    //AV related
    BOOL bAVEnabled=TRUE;
    DWORD gdwAVProductID=1001;
    CMPVERS gcmpAVVersion;
    BOOL bAVAutoUpdate=TRUE;
    BOOL bAVRealTimeMonitor=TRUE;
    // router info
    CMPVERS gcmpRouteVersion;
    DWORD gdwServerAddr;//192.168.0.1
    DWORD gdwRouteSessionID=0;
    // return from client and to be saved in client table
    DWORD gdwclientStatus, gdwClientIP=0;
    // when authorize connection
    // redirect to sandbox
    DWORD gdwDstIPAddr=0;
    WORD gwPortDst=80;
    // number of licences being assigned
    WORD gwNumberOfUsers=0;
    int iIndex;
    DWORD atReturn;
    // socket, subnet address
    SOCKET socketCMP;
    int iTimeout=5000;  //receive time out
    //packet space
```

```
  char packetInOut[CMP_MAX_CHALLENGE], packetRSA[CMP_MAX_CHALLENGE];
DWORD tcStart;
DWORD tcNow;
  DWORD dwPresentTime;
  BOOL rStatus;
  DWORD dwLenReturned;
// hard code ZAP version
gcmpZAPVersion.wVers[0]=2;
gcmpZAPVersion.wVers[1]=6;
gcmpZAPVersion.wVers[2]=0;
gcmpZAPVersion.wVers[3]=88;
// hard code AV version
gcmpAVVersion.wVers[0]=2;
gcmpAVVersion.wVers[1]=6;
gcmpAVVersion.wVers[2]=6;
gcmpAVVersion.wVers[3]=6;
// hard code router version
gcmpRouteVersion.wVers[0]=2;
gcmpRouteVersion.wVers[1]=6;
gcmpRouteVersion.wVers[2]=8;
gcmpRouteVersion.wVers[3]=6;
// clean buffer
memset(packetInOut,0,CMP_MAX_CHALLENGE);
memset(packetRSA,0,CMP_MAX_CHALLENGE);
// set event for exiting loop
  if(FALSE==SetConsoleCtrlHandler((PHANDLER_ROUTINE)CtrlHandler,TRUE))
     return;
//gdwServerAddr=0xc0a80001;//192.168.0.1
  gdwServerAddr=0x7f000001; // 127.0.0.1
  gdwServerAddr=0xac1014a5; // 172.16.20.165
gdwServerAddr=0xc0a8326f; // 192.168.50.111
  // Initialize client table
InitializeGlobals(gdwServerAddr);
// swap IP address
gdwServerAddr=SwapDWord(gdwServerAddr);
  if(CMPGetSocket(&socketCMP, gdwServerAddr, iTimeout)==SOCKET_ERROR)
 return;
  // check command line arguments
if ( argc>1 && (( *argv[1] == '-' ) || ( *argv[1] == '/' )) )
{
 //talk with client through challenge/hello/response
   if(*(argv[1]+1)=='c' || *(argv[1]+1)=='C' ||
   *(argv[1]+1)=='t' || *(argv[1]+1)=='T' ||
   *(argv[1]+1)=='a' || *(argv[1]+1)=='A' ||
   *(argv[1]+1)=='l' || *(argv[1]+1)=='L' ||
   *(argv[1]+1)=='v' || *(argv[1]+1)=='V' ||
   *(argv[1]+1)=='s' || *(argv[1]+1)=='S' ||
   *(argv[1]+1)=='p' || *(argv[1]+1)=='P')
 {
  dwLenReturned=0;
```

```
   // first packet is a control panel setting packet
   //talk with client through challenge/hello/response
   //!!!!!!!!!!!!!!!
   // the change should be reflected in every heartbeat
   // please pick up code for different combinations
   //!!!!!!!!!!!!!!!
   // ************************************** challenge + any option starts
   if(*(argv[1]+1)=='s' || *(argv[1]+1)=='S')//send control panel setting, not tested
   {
    // ZAP related
    //new check time, response time=check time/2
    //new download site, default linksys.zonelabs.com
    //new user prompt
    //new required client version
    //new license key
    //new av related
    WORD wTempPacketSize;
    PRT_CHALLENGE_PACKET pTempChallenge;
    bSettings=FALSE;
    bSettings=CMPCreateChallengePacket(packetInOut, CMP_MAX_CHALLENGE, CMPP_LINKSYS, gdwServerAddr,
      (UCHAR*)&gcmpRouteVersion, gdwRouteSessionID, gdwResponseTime, GetTickCount()/1000);
    pTempChallenge=(PRT_CHALLENGE_PACKET)packetInOut;
    //ZAP related
    if(bZAPEnabled)
    {
/*****************************************/
// UserPrompt sample code begins
#if DO_USERPROMPT
     // user prompt packet if changed
     if(bSettings)
     {
      wTempPacketSize=pTempChallenge->wPacketSize;
      bSettings=CMPAddUserPromptOption(packetInOut, CMP_MAX_CHALLENGE, newUserPrompt);
      SwapOptUserPrompt((POPT_USER_PROMPT)((UCHAR*)packetInOut+wTempPacketSize));
     }
#endif
// UserPrompt sample code ends
/*****************************************/
     // licence issued
     if(bSettings)
     {
      wTempPacketSize=pTempChallenge->wPacketSize;
      bSettings=CMPAddLicenseOption(packetInOut, CMP_MAX_CHALLENGE, gwNumberOfUsers, newLicenseKey);
      SwapOptLicense((POPT_ZL_LICENSE)((UCHAR*)packetInOut+wTempPacketSize));
     }
     // version packet if changed
     if(bSettings)
     {
      wTempPacketSize=pTempChallenge->wPacketSize;
      bSettings=CMPAddClientVersionOption(packetInOut, CMP_MAX_CHALLENGE, CMPP_ZAPRO,
(UCHAR*)&gcmpZAPVersion);
```

```
     SwapOptClientVersion((POPT_CLIENT_VERSION)((UCHAR*)packetInOut+wTempPacketSize));
    }
   }
/*******************************************/
// AntiVirus sample code begins
#if DO_ANTIVIRUS
   // AV related
   if(bAVEnabled)
   {
    // AV info if changed
    if(bSettings)
    {
     wTempPacketSize=pTempChallenge->wPacketSize;
     bSettings=CMPAddAntivirusOption(packetInOut, CMP_MAX_CHALLENGE, gdwAVProductID,
(UCHAR*)&gcmpAVVersion,
            bAVAutoUpdate, bAVRealTimeMonitor, 0, 0);
     SwapOptAntiVirus((POPT_ANTI_VIRUS)((UCHAR*)packetInOut+wTempPacketSize));
    }
   }
#endif
// AntiVirus sample code ends
/*******************************************/
   // checksum and encrypt
   if(bSettings)
   {
    wTempPacketSize=pTempChallenge->wPacketSize;
    SwapRouterChallenge((PRT_CHALLENGE_PACKET)packetInOut);
    // checksum
    pTempChallenge->dPacketCRC=CheckSum((WORD*)packetInOut, wTempPacketSize, 0);
    // ecrypt the packet
    dwLenReturned=CMPEncryptPacket(packetInOut, wTempPacketSize, packetRSA, CMP_MAX_CHALLENGE);
   }
  }//!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
  // *************************************** challenge + any option ends
/*******************************************/
// UserPrompt sample code begins
#if DO_USERPROMPT
        // first packet is a user prompt packet
  //talk with client through challenge/hello/response
     if(*(argv[1]+1)=='p' || *(argv[1]+1)=='P')
  {
   if(argc>2)
   {
    if(strlen(argv[2])<60)
     strcpy(cUserPrompt,argv[2]);
   }
   else
    strcpy(cUserPrompt,newUserPrompt);
   dwLenReturned=PrepareUserPromptPacket(packetInOut, packetRSA,
       gdwServerAddr, (UCHAR*)&gcmpRouteVersion, gdwRouteSessionID,
```

```
        gdwResponseTime, cUserPrompt);
    }
#endif
// UserPrompt sample code ends
/*****************************************/
    // first packet is aversion packet
    //talk with client through challenge/hello/response
      if(*(argv[1]+1)=='v' || *(argv[1]+1)=='V')
    {
     if(argc>2)
     {//201
                  if(ParseVersion(argv[2], &gcmpZAPVersion)==FALSE)
                  {
                   gcmpZAPVersion.wVers[0]=2;
                   gcmpZAPVersion.wVers[1]=7;
                   gcmpZAPVersion.wVers[2]=0;
                   gcmpZAPVersion.wVers[3]=0;
                  }
     }
     dwLenReturned=PrepareVersionInquiryPacket(packetInOut, packetRSA,
        gdwServerAddr, (UCHAR*)&gcmpRouteVersion, gdwRouteSessionID,
        gdwResponseTime,(UCHAR*)&gcmpZAPVersion);
    }
    // first packet is license key packet
    //talk with client through challenge/hello/response
      if(*(argv[1]+1)=='l' || *(argv[1]+1)=='L')
    {
     strcpy(cUserLicense,newLicenseKey);
     if(argc>2)
     {//201
      if(strlen(argv[2])==27 && CMPValidateLicenseKey(argv[2]))
       strcpy(cUserLicense,argv[2]);
      else
       strcpy(cUserLicense,newLicenseKey);
     }
     dwLenReturned=PrepareLicenseKeyPacket(packetInOut, packetRSA,
         gdwServerAddr, (UCHAR*)&gcmpRouteVersion, gdwRouteSessionID,
         gdwResponseTime, gwNumberOfUsers, cUserLicense);
    }
/*****************************************/
// AntiVirus sample code begins
#if DO_ANTIVIRUS
    // first packet is AV inquiry packet
    //talk with client through challenge/hello/response
      if(*(argv[1]+1)=='a' || *(argv[1]+1)=='A')
    {
     dwLenReturned=PrepareAVInquiryPacket(packetInOut, packetRSA,
      gdwServerAddr, (UCHAR*)&gcmpRouteVersion, gdwRouteSessionID, gdwResponseTime,
      gdwAVProductID, (UCHAR*)&gcmpAVVersion, bAVAutoUpdate, bAVRealTimeMonitor);
    }
```

```
#endif
// AntiVirus sample code ends
/*****************************************/
  //talk with client through challenge/hello/response
  if(*(argv[1]+1)=='c' || *(argv[1]+1)=='C' ||
          *(argv[1]+1)=='t' || *(argv[1]+1)=='T')
   dwLenReturned=PrepareHeartBeatPacket(packetInOut, packetRSA,
    gdwServerAddr, (UCHAR*)&gcmpRouteVersion, gdwRouteSessionID, gdwResponseTime);
  if(dwLenReturned)
   CMPBroadcast(socketCMP, packetRSA, dwLenReturned, gdwServerAddr);
  // to get the current tick to control the heartbeat frequency
  // ************************************************* main loop starts
  // ************************************************* main loop starts
  tcStart = GetTickCount();
  while (wLoopControl)
  {
   // ****************** periodical listen and analysis starts
   if(dwLenReturned=CMPListenToClient(socketCMP, gdwServerAddr, packetInOut, CMP_MAX_CHALLENGE,
&gdwClientIP))
   {
    // after receiving a packet successfully
    if(dwLenReturned=CMPDecryptPacket(packetInOut, dwLenReturned, packetRSA, CMP_MAX_CHALLENGE))
    {
     if(rStatus=CMPAnalyseDecryptedPacket(packetRSA, dwLenReturned, &gdwclientStatus))
     {
      if(rStatus==CMPM_CLIENT_HELLO)
      {
       // send response to heart beat immediately with response time to 0
       dwLenReturned=PrepareHeartBeatPacket(packetInOut, packetRSA,
        gdwServerAddr, (UCHAR*)&gcmpRouteVersion, gdwRouteSessionID, 0);
       if(dwLenReturned)
        CMPBroadcast(socketCMP, packetRSA, dwLenReturned, gdwClientIP);
                        printf("======Received Hello======<IP address=%d.%d.%d.%d>",
                         (gdwClientIP & 0x000000ff),(gdwClientIP & 0x0000ff00)>>8,
                         (gdwClientIP & 0x00ff0000)>>16, (gdwClientIP & 0xff000000)>>24);
      }
      else if(rStatus==CMPM_CLIENT_GOODBYE)//update sample client table
                      {
       iIndex=SwapDWord(gdwClientIP)-SwapDWord(tGlobalClient[0].dwIPAddr);
                        //the status includes CMP and license status
                        /*****************************************/
       tGlobalClient[iIndex].dwStatus=CMPR_CLIENT_TIMEOUT;
                        printf("======Received GoodBye======<IP address=%d.%d.%d.%d>",
                         (gdwClientIP & 0x000000ff),(gdwClientIP & 0x0000ff00)>>8,
                         (gdwClientIP & 0x00ff0000)>>16, (gdwClientIP & 0xff000000)>>24);
                      }
      else if(rStatus==CMPM_CLIENT_RESPONSE)//update sample client table
      {
       iIndex=SwapDWord(gdwClientIP)-SwapDWord(tGlobalClient[0].dwIPAddr);
                        //the status includes CMP and license status
```

```
                    /*****************************************/
    tGlobalClient[iIndex].dwStatus=gdwclientStatus;
    PrintClientStatus("======Received Response======");
   }
  }
 }
}
// ******************* periodical listen and analysis ends
// ********heartbeat******** starts
// to get the current tick to control the heartbeat frequency
  tcNow = GetTickCount();
// heartbeat time in seconds on ?
if (tcNow-tcStart >= (gdwResponseTime * 1000))
{
 tcStart = tcNow;
 // get number of licenses being accepted
 gwNumberOfUsers=CMPCountLicenses(gdwResponseTime*4);
 if(*(argv[1]+1)=='v' || *(argv[1]+1)=='V')
 {
  // version heartbeat
  dwLenReturned=PrepareVersionInquiryPacket(packetInOut, packetRSA,
      gdwServerAddr, (UCHAR*)&gcmpRouteVersion, gdwRouteSessionID,
      gdwResponseTime,(UCHAR*)&gcmpZAPVersion);
 }
 else if(*(argv[1]+1)=='l' || *(argv[1]+1)=='L')
 {
  // license heartbeat
  dwLenReturned=PrepareLicenseKeyPacket(packetInOut, packetRSA,
       gdwServerAddr, (UCHAR*)&gcmpRouteVersion, gdwRouteSessionID,
       gdwResponseTime, gwNumberOfUsers, cUserLicense);
 }
 else
 {
  // challenge heartbeat
  dwLenReturned=PrepareHeartBeatPacket(packetInOut, packetRSA,
   gdwServerAddr, (UCHAR*)&gcmpRouteVersion, gdwRouteSessionID, gdwResponseTime);
 }
 // send out heartbeat every 30 seconds
 if(dwLenReturned)
  CMPBroadcast(socketCMP, packetRSA, dwLenReturned, gdwServerAddr);
 printf(".");
}
// ********heartbeat******** ends
// ******************** demo of checking traffic periodically starts
if((wLoopControl%10) == 9)//authorize traffic, not tested
{
 char stringIP[28];
 gdwDstIPAddr=SwapDWord(gdwClientIP);
 sprintf(stringIP,"%d.%d.%d.%d",(gdwDstIPAddr & 0xff000000)>>24,(gdwDstIPAddr & 0x00ff0000)>>16,
       (gdwDstIPAddr & 0x0000ff00)>>8, gdwDstIPAddr & 0x000000ff);
```

```
    if( (*(argv[1]+1)=='t' || *(argv[1]+1)=='T'))
    {
     gdwDstIPAddr=0;
     if(argc>2)
     {
      // string to IP address
      gdwDstIPAddr=StringToAddress(argv[2]);
      if(gdwDstIPAddr==0)
       printf("\n Wrong IP address <%s> inputed !! sample <192.66.55.218>",argv[2]);
      else
       strcpy(stringIP,argv[2]);
     }
     else
      printf("\n Please input IP address,format: router /t 192.66.55.218");
    }
    if(gdwDstIPAddr)
    {
     // when to connect to the internet
     dwPresentTime=GetTickCount()/1000;
     // presume IP address is in a continuous range starting from 192.168.0.0
     iIndex=gdwDstIPAddr-SwapDWord(tGlobalClient[0].dwIPAddr);
     if(iIndex<0 || iIndex>255)
      printf("\n Not router related IP address <%s> inputed !!",stringIP);
     else
     {
      atReturn=CMPAuthorizeTraffic(gwPortDst, tGlobalClient[iIndex].dwStatus & CMPR_MASK, dwPresentTime,
gdwResponseTime*4);
      if(atReturn==0)
       printf("\nThe traffic request by %s is authorized to pass !",stringIP);  //pass
      else if(atReturn>0)//redirect to LinkSys.ZoneLabs.com, port number = atReturn
       printf("\nThe traffic request by %s is directed to Linksys.ZoneLabs.com:%d",stringIP,atReturn);
      else
       printf("\nThe traffic request by %s is blocked !",stringIP);  //block
     }
    }
   }
   wLoopControl++;
   // ********** demo of checking traffic periodically ends
  }
  // ************************************************** main loop ends
  // ************************************************** main loop ends
 }
}
   else
   {
      printf("\n *To end the heartbeat, type CTRL-C/CTRL-BREAK*");
      printf("\n ******** To test heartbeat ********");
      printf("\n router /p prompt string");
      printf("\n example: router /p ZAP is enforced to run");
      printf("\n router /v version number");
```

```
        printf("\n example : router /v 2.6.0.300");
        printf("\n router /l license string");
        printf("\n example: router /l abcdefghijklmnopqrstuvwxyz1");
        printf("\n router /s");
        printf("\n default settings plus heartbeat");
        printf("\n router /c");
        printf("\n heartbeat");
        printf("\n router /a");
        printf("\n example: not available");
        printf("\n ******** To test authorizing traffic ********");
        printf("\n router /t IP address");
        printf("\n example: router /t 192.66.55.218");
    }
    CMPKillSocket(socketCMP);
}
```